ATTORNEY DOCKET NO.

NVIDP074/P000427

U.S. PATENT APPLICATION

FOR

SYSTEM AND METHOD FOR CALCULATING PARTIAL DIFFERENTIAL EQUATIONS IN A HARDWARE GRAPHICS PIPELINE

ASSIGNEE: *n*VIDIA CORPORATION

KEVIN J. ZILKA
SILICON VALLEY IP GROUP
P.O. BOX 721120
SAN JOSE, CA 95172

SYSTEM AND METHOD FOR CALCULATING PARTIAL DIFFERENTIAL EQUATIONS IN A HARDWARE GRAPHICS PIPELINE

FIELD OF THE INVENTION

The present invention relates to computer graphics, and more particularly, to calculating partial differential equations in a computer graphics processing pipeline.

BACKGROUND OF THE INVENTION

Three dimensional graphics are central to many applications. For example, computer aided design (CAD) has spurred growth in many industries where computer terminals, cursors, CRT's and graphics terminals are replacing pencil and paper, and computer disks and tapes are replacing drawing vaults. Most, if not all, of these industries have a great need to manipulate and display three-dimensional objects. This has lead to widespread interest and research into methods of modeling, rendering, and displaying three-dimensional objects on a computer screen or other display device. The amount of computations needed to realistically render and display a three-dimensional graphical object, however, remains quite large and true realistic display of three-dimensional objects have largely been limited to high end systems. There is, however, an ever-increasing need for inexpensive systems that can quickly and realistically render and display three dimensional objects.

One industry that has seen a tremendous amount of growth in the last few years is the computer game industry. The current generation of computer games is moving to three-dimensional graphics in an ever increasing fashion. At the same time, the speed of play is being driven faster and faster. This combination has fueled a genuine need for the rapid rendering of three-dimensional graphics in relatively inexpensive systems. In addition to gaming, this need is also fueled by e-Commerce applications, which demand increased multimedia capabilities.

Rendering and displaying three-dimensional graphics typically involves many calculations and computations. For example, to render a three dimensional object, a set of coordinate points or vertices that define the object to be rendered must be formed. Vertices can be joined to form polygons that define the surface of the object to be rendered and displayed. Once the vertices that define an object are formed, the vertices must be transformed from an object or model frame of reference to a world frame of reference and finally to two-dimensional coordinates that can be displayed on a flat display device. Along the way, vertices may be rotated, scaled, eliminated or clipped because they fall outside the viewable area, lit by various lighting schemes, colorized, and so forth. Thus the process of rendering and displaying a three-dimensional object can be computationally intensive and may involve a large number of vertices.

One specific operation that occurs in an application program in addition to rendering is the calculation of partial differential equations. For example, partial differential equations can be used to determine the location of objects or surfaces to be rendered. Such partial differential equations are traditionally calculated utilizing software and a central processor unit (CPU), since such calculations are generally not cost effective to be implemented in hardware. However, calculating partial differential equations in software places a heavy burden of computation on the system which may be a significant bottleneck in the application program, and can therefore impede the process of rendering. With the growing demand for higher performance graphics for computers, the need for speeding up the solution of partial differential equations becomes imperative.

There is thus a general need to increase the speed and efficiency of the various processing components, while minimizing costs. In general, consolidation is often employed to increase the speed of a system. Consolidation refers to the incorporation of different processing modules on a single integrated circuit. With such processing modules communicating in a microscopic semiconductor environment, as opposed to external buses, speed and efficiency may be vastly increased. Additional gain may be achieved by performing many operations in parallel, as is common in a hardware graphics pipeline.

Consolidation is often limited, however, by a cost of implementing and manufacturing multiple processing modules on a single chip. One reason for this increase in cost is that the required integrated circuit would be of a size that is too expensive to be feasible. Rather than add unique circuitry to handle a problem, it is advantageous to leverage and extend existing circuitry in novel ways to handle the problem.

SUMMARY OF THE INVENTION

A system and method are provided for computing partial differential equations in a hardware graphics pipeline. Initially, input is received in a hardware graphics pipeline. Next, the input is processed to generate a solution to a partial differential equation utilizing the hardware graphics pipeline.

In one embodiment, the input may represent boundary conditions. This may be accomplished by the input including textures or geometry. As an option, the geometry may include polygons, vertex data, points, and/or lines.

In another embodiment, the input may include a local area of textures. Such local area of textures may be generated by sampling a texture map. Further, the local area of textures may be filtered. This may be accomplished utilizing a plurality of programmable and/or non-programmable filters. Moreover, such filters may include a plurality of elements. Optionally, the local area of textures may be used to sample a texture map to generate a modified local area of textures.

In still another embodiment, the processing may include a relaxation operation. Such relaxation operation may be selected based on the particularities of the partial differential equation. Moreover, the processing may include a plurality of iterations of the relaxation operation. Such number of iterations of the relaxation operation may be reduced using a combination of restriction and prolongation operations. Further, such number of iterations of the relaxation operation may be calculated prior to the processing or via a real-time convergence technique.

In an embodiment using the real-time convergence technique, the processing may further include determining whether the solution has converged. It may be determined whether the solution has converged after each iteration of the relaxation operation. In the alternative, it may be determined whether the solution has converged after a predetermined number of multiple iterations of the relaxation operation.

As an option, it may be determined whether the solution has converged by calculating errors. Such errors may then be summed or evaluated by various techniques after which it may be concluded that the solution has converged if, for example, the sum of errors or the greatest error is less than a predetermined amount. If it is determined that the solution has converged, the processing may be repeated using an incremented parameter value of the equation, for example a time value.

Brief Description of the Drawings

The foregoing and other aspects and advantages are better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, in which:

Figure **1A** is a block diagram of a digital processing system.

Figure **1B** illustrates the hardware graphics pipeline of Figure **1A**, in accordance with one embodiment.

Figure **1C** shows an illustrative hardware implementation of the pixel processor portion of the hardware graphics pipeline of Figure **1A**, in accordance with one exemplary embodiment.

Figure **2A** illustrates a method for computing partial differential equations in a hardware graphics pipeline.

Figure **2B** illustrates a method for determining whether a solution has converged, in accordance with decision **206** of Figure **2A**.

Figure **3** shows the basic components used in calculating partial differential equation solutions in a hardware graphics pipeline.

Figure **4** presents the steps for a brute force solution using the relaxation method.

Figure **5A** demonstrates the combination of texture and geometry data in rendering one step of the solution.

Figure **5B-D** illustrates various techniques associated with the multi-sampling aspect set forth in Figure **5A**.

Figure **6** illustrates the sampling and processing for a rendered pixel during one step of the solution.

Figure **7A** further illustrates the sampling and processing for a rendered pixel.

Figure **7B** illustrates a technique by which initial samples may control the addresses from which subsequent samples are taken

Figure **8** illustrates a method for the calculation and summation of residuals or errors via rendering operations.

Figure **9** illustrates a technique for the implementation of a multi-grid relaxation method solution in a hardware graphics pipeline.

Figure **10** illustrates a few grids of various resolutions as may be used in a multi-grid approach.

Figure **11** illustrates a simple sampling pattern for prolongation and restriction as implemented in a hardware graphics pipeline.

DETAILED DESCRIPTION

Figure **1A** is a block diagram of a digital processing system embodying the method and apparatus in accordance with one embodiment. With reference to Figure **1A**, a computer graphics system is provided that may be implemented using a computer **1400**. The computer **1400** includes one or more processors, such as processor **1401**, which is connected to a communication bus **1402**. The computer **1400** also includes a main memory **1404**. Control logic (software) and data are stored in the main memory **1404** which may take the form of random access memory (RAM). The computer also includes a hardware graphics pipeline **1406** and a display **1408**, i.e. a computer monitor.

The computer **1400** may also include a secondary storage **1410**. The secondary storage **1410** includes, for example, a hard disk drive and/or a removable storage drive, representing a floppy disk drive, a magnetic tape drive, a compact disk drive, etc. Computer programs, or computer control logic algorithms, are stored in the main memory **1404** and/or the secondary storage **1410**. Such computer programs, when executed, enable the computer **1400** to perform various functions. Memory **1404** and storage **1410** are thus examples of computer-readable media.

In one embodiment, the techniques to be set forth are performed by the hardware graphics pipeline **1406** which may take the form of any type of hardware. Such hardware implementation may include a micro-controller or any other type of application specific integrated circuit (ASIC). More information on one exemplary embodiment of the hardware graphics pipeline **1406** will be set forth in greater detail during reference to Figure **1B**. The configuration of the graphics pipeline and general control over rendering operations is provided by the processor **1401** which may also prepare appropriate geometry and basis values.

Figure **1B** illustrates the hardware graphics pipeline **1406** of Figure **1A**, in accordance with one embodiment. As shown, the hardware graphics pipeline 1406 may include a vertex processor portion **1500** capable of performing various operations (i.e. transformation, lighting, etc.) on graphics data in the form of vertices in the hardware graphics pipeline **1406.** Further included is a rasterizer **1502** coupled to the vertex processor portion **1500** for receiving the processed vertices therefrom. Such vertices define primitives. In the context of the present description, primitives may include, but are not limited to planar polygons. Alternate graphics primitives, for alternate embodiments, include: points, lines, quadratic patches, constructive solid geometry surfaces, and other higher order primitives. In use, the rasterizer **1502** converts the primitives to fragments that correspond to a pixel in a frame buffer render target. Further, a digital-to-analog converter (DAC) **1505** and pipeline memory **1501** may be incorporated in a well known manner.

Once rasterized, the fragments are processed by a pixel processor portion **1504** which is coupled to the rasterizer **1502.** Despite the term "pixel processor" **1504**, it should be noted that such portion processes texture fragments and/or pixels. Such processing may include, but is not limited to texturing, shading, texture coordinate perturbation, etc. More information on one exemplary embodiment of the pixel processor portion **1504** will be set forth in greater detail during reference to Figure **1C**.

Of course, any of the foregoing components of the hardware graphics pipeline **1406** may or may not be configurable in any desired fashion. Further, the various techniques to be described herein may be implemented in any one or more of the components of the hardware graphics pipeline **1406**, per the desires of the user.

Figure **1C** shows an illustrative hardware implementation of the pixel processor portion **1504**, in accordance with one exemplary embodiment. As shown, included is a shader module **1516** coupled to the rasterizer **1502**, a texture fetch module **1518**, and a combiner **1525** coupled to form a portion of the hardware graphics pipeline **1406.** For reasons that will soon become apparent, a feedback loop **1519** is coupled between an output of the shader module **1516** and an input thereof. It should be noted that the rasterizer **1502** operates in a manner as set forth during reference to Figure **1C**. While the combiner **1525** may be implemented in any desired manner, one exemplary implementation is disclosed in a co-pending application entitled "IMPROVED GRAPHICS PIPELINE INCLUDING COMBINER STAGES" filed 03/20/99 naming David B. Kirk, Matthew Papakipos, Shaun Ho, Walter Donovan, and Curtis Priem as inventors, and which is incorporated herein by reference in its entirety.

With continuing reference to Figure **1C**, the various inputs and outputs are shown for each of the components. The rasterizer **1502** generates fragment data (i.e. interpolated vertex data, edge distances, pixel depth) which are used by the shader module **1516** and texture fetch module **1518**. Also shown is an optional feedback first-in first-out (FIFO) buffer. When the feedback loop **1519** is not utilized, the temporary data calculated internally by the present embodiment may be dropped before being sent to the texture fetch module **1518**. As an option, data may be reused each time a particular group of pixels, or "quad," goes through the shader module **1516**. If, for example, new vectors are generated during one pass, these vectors may continuously be associated with the quad on subsequent passes. Further, more than one fragment may be processed at a time while employing the feedback loop **1519**.

A system and method may thus be provided for computing partial differential equations in the foregoing hardware graphics pipeline. Initially, input is received in the hardware graphics pipeline. Next, the input is processed to generate a solution to a partial differential equation utilizing the hardware graphics pipeline. More information regarding a specific implementation of such technique will be set forth during reference to the following figures.

Figure **2A** illustrates a method **200** for computing partial differential equations in a hardware graphics pipeline. In one embodiment, the present method **200** may be carried out in the context of the architecture of at least one of Figures **1A-1C**. Of course, the method **200** may be implemented in any desired architecture capable of carrying out the functionality set forth herein.

Initially, boundary conditions are received in operation **202**. These boundary conditions may take the form of textures or geometry. As an option, the geometry may include polygons, vertex data, points, and/or lines. More information on the form of the input will be set forth in greater detail during reference to Figure **3**.

Next, a solution to a partial differential equation is calculated in operation **204** utilizing a relaxation operation involving the boundary conditions and the discreet grid of values representing the state of the equation. As an option, such relaxation operation may be selected based on the partial differential equation. In the context of the present description, a relaxation operation may include the filtering or smoothing of grid values._More information regarding an exemplary relaxation operation will be set forth during reference to Figures **4 - 6**.

As an option, the boundary conditions may take the form of a local area of textures. Such local area of textures may be generated by sampling a texture map. Further, the local area of textures may be filtered. This may be accomplished utilizing a plurality of programmable and/or non-programmable filters. Moreover, such filters may include a plurality of elements. More information regarding an example of such filtering will be set forth hereinafter during reference to Figure **7A**. Optionally, the local area of textures may be used to sample a texture map to generate a modified local area of textures, as will described in greater detail in the context of an example during reference to Figure **7B**.

In use, the relaxation operation may be repeated during the course of multiple iterations. Such number of iterations of the relaxation operation may be calculated prior to the processing or via a real-time convergence technique. In the context of the present method **200**, a convergence technique is utilized.

It is then determined, in decision **206**, whether the solution has converged. If the solution has not converged, as determined by decision **206**, the computing and determining operations (**204** and **206**, respectively) are repeated. See line **207**. In an alternate embodiment, it may be determined whether the solution has converged after a predetermined number of multiple iterations of the relaxation operation, instead of after each iteration. If the solution has converged in decision **206**, the solution may be utilized in operation **208**. More information as to how it is determined whether a solution has converged will be set forth during reference to Figure **2B**.

As an option, the number of iterations of the relaxation operation may be reduced using a combination of restriction and prolongation operations. In the context of the present description, a restriction operation may include the mapping and filtering of grid values onto a grid of lower resolution, and a prolongation operation may include the mapping and filtering of grid values onto a grid of higher resolution.

Next, one or more parameters of the partial differential equation may be adjusted in operation **210**. Such parameters may include a time value or any other suitable parameter of the equation. It may then be determined whether the present method is finished in decision **212**. If so, the foregoing operations may be repeated using the incremented parameter values, as shown. If not, however, the present method **200** is terminated. At any step of the method, the state of the equation may be visualized by the user. Such visualization may be rapid and inexpensive as the state is held within the hardware graphics pipeline.

Figure **2B** illustrates a method for determining whether a solution has converged, in accordance with decision **206** of Figure **2A**. While one particular method is being set forth herein for determining whether the solution converged, any particular method may be employed per the desires of the user.

Initially, a plurality of errors are calculated in operation **252** after which they are summed in operation **254**. If the sum of errors is less than a predetermined amount per decision **255**, it is concluded in operation **256** that the solution has converged. More information relating to such error calculation will be set forth hereinafter during reference to Figure **8**.

Figure **3** illustrates basic computer graphics elements **300** which may be employed when solving partial differential equations while rendering in a hardware graphics pipeline, in accordance with operation **202** of Figure **2A**. As shown, geometry **301**, generally a simple object to cover all pixels of a render target surface, is used to initiate processing of texture elements **302** which contain the state of the solution as the method progresses. This geometry **301** may include, but is not limited to polygons, points, lines, and vertex data with position, texture, color, etc.

One or more texture maps may hold the boundary conditions and values **305**, **307** which govern the solution of the partial differential equations. Boundary conditions and values may also be represented by geometry **306**, **308** of various shape and size. Both periodic and non-periodic boundary conditions may be thus supplied, and the texture wrapping mode (whether textures are tiled continuously, clamped to border values, mirrored, etc.) may determine the nature of the boundary conditions. For example, as illustrated by boundary values **307,** a texture may enforce boundary values along the upper and lower edges and not enforce boundary values along the left and right edges. Combined with horizontal texture tiling, this may give rise to a boundary condition periodic in the horizontal axis and non-periodic in the vertical axis.

Figure **4** illustrates two variations **402**, **404** of a brute force relaxation operation for generating the solution of a partial differential equation, in accordance with operation **204** of Figure **2A**.

As shown in the first variation **402**, the partial differential equation is initially discretized to a uniform grid **406** of a given mesh size. Further, a relaxation operation is performed in operation **407**. Such technique is well known to those of ordinary skill. For example, an abstract description and derivation of these methods out of the present context may be found with reference to: Press, William H., Teukolsky, Saul A., et. al., "Numerical Recipes in C 2nd. Ed.", Cambridge University Press, 1992, which is incorporated herein by reference in its entirety. Finally, the solution may be evaluated for convergence in operation **408** after each or a number of relaxation steps.

In the alternate variation **404**, a calculation or estimate **410** of the number of relaxation steps, *n*, required for convergence may be carried out before relaxation begins. The relaxation operation steps may then be carried out *n* times to achieve a solution, which may also be evaluated for convergence.

Figure **5A** illustrates the use of geometry **500** in rendering a collection of input stored in textures **502**, **503** to destination render targets **504**, in accordance with operation **204** of Figure **2A**. In one embodiment, the present technique may establish a one-to-one correspondence between texels of a source textures **502** to pixels of the render targets **504**, such that when the geometry **500** is rendered, an exact copies of the source textures **502** are rendered into the render targets **504**.

Expanding on this approach in an unillustrated alternate embodiment, several textures **502** may be input and, in the process of rendering, their samples may be processed in various ways as described below. In such embodiment, texture coordinates may be used to determine which samples of the source textures **502** are supplied to a shading unit during rendering. By expressing these texture coordinates as offsets from a one-to-one mapping, a local area of texels may be supplied surrounding each pixel. In this way, it may be specified that each pixel rendered samples a given pattern of neighbors from the source textures **502**. For example, each pixel may be supplied with its four nearest neighbors or any other pattern and number of neighbors. More information on such neighbor sampling will now be set forth in greater detail.

Figure **5B** depicts the steps involved in the configuration **5200** of the hardware graphics pipeline for neighbor sampling. Texture and shader data is bound to the pipeline in step **5201**. Sampling the required basis values is specified **5202**, as well as the method **5203** by which the basis values are to be processed to yield vectors. In the context of the present description, the samples from which data is derived are referred to as basis values. The pipeline is also directed in operation **5204** to output data to one or more render targets.

Figure **5C** depicts the steps carried out by the graphics pipeline in the act of rendering according to the configuration **5200** established previously. It also depicts an optional feedback loop **5310** whereby calculated values may be used to control additional sampling of basis values. This rendering writes visible pixels to a frame buffer as in operation **5304** and/or writes data to a texture or other render target as in operation **5305**. Figure **5D** depicts the basic rendering loop in which the hardware graphics pipeline is configured and rendering progresses. Each circuit through this loop may render all or part of a scene including intermediate steps not visible in the final rendered image.

More information regarding such neighbor sampling may be found with reference to an application filed coincidently herewith naming inventors Greg Edward James and Matthias Wloka under the title: "SYSTEM AND METHOD FOR CREATING A VECTOR MAP IN A HARDWARE GRAPHICS PIPELINE" and docket number NVIDP072/P000426, which is incorporated herein by reference in its entirety.

Figure **6** illustrates the calculation **600** of one pixel grid value in the solution during one step of a relaxation operation as carried out in a hardware graphics pipeline, in accordance with operation **204** of Figure **2A** or more specifically operation **407** of Figure **4**. The underlying geometry has been omitted from this diagram and subsequent diagrams for clarity purposes.

As shown in Figure **6**, each rendered pixel **602** samples a local area **604** of texels **606**. This local area **604** is processed according to a relaxation operation to yield a 'relaxed' value which is written to the destination. The nature of the relaxation operation may be governed by the formulation of the partial differential equation being solved. A specific example of this will be given hereinafter. Carried out for every pixel of the discreet grid representing the state of the equation, the calculation **600** of Figure **6** constitutes one relaxation step **407** as shown in Figure **4.** By carrying out many of these relaxation operations, supplying each relaxed grid as input to the subsequent relaxation step via the feedback **605**, the grid of values may converge to the solution of the partial differential equation.

Figure **7A** elaborates on the sampling of Figure **6** to show various convolution kernels or filters **700** that may be applied to the local area **604** of samples in generating various intermediate fragment values **701**. Such fragment values are processed to yield various pixel values **602**. As shown, programmable or non-programmable filters **700** including one or more elements **702** may be used. In one embodiment, three elements by three elements (3x3) may be used. In practice, such filters **700** may contain any number of elements **702**, and a plurality of filters **700** may be employed in the rendering.

Figure **7B** illustrates a technique **703** by which intermediate values **701** as computed in Figure **7A** may also affect the coordinates at which subsequent values **706** are sampled. Any intermediate values **701** or combinations thereof may then be used to determine the coordinates of sampling via feedback **708**. Such a feedback mechanism may be useful in varying the pattern of sampling from one relaxation operation to the next. Such variation may reduce the number of relaxation steps required, and provides a basis in the context of a hardware graphics pipeline for the mathematical red/black checkerboard technique of relaxation. Abstract information on such technique may be found out of this context in the aforementioned reference [Press].

Figure **8** illustrates a technique **800** by which errors **802** (i.e. residuals) may be calculated and summed, averaged, or otherwise reduced to a smaller set of error values over subsequent rendering operations in a hardware graphics pipeline, in accordance with the various operations of Figure **2B**. As set forth earlier, such evaluation of errors **802** and reduction **804** may be used in determining the convergence and accuracy of a solution and may be employed at any time for such purpose.

This technique may be carried out in a manner similar to the well known method of creating texture mipmaps in a hardware graphics pipeline. The creation of mipmaps involves an average of local areas, however, a sum or more advanced calculation may be performed in the reduction to yield a single error metric.

Possibly, such an accumulation of errors may only be meaningful if carried out with floating point texture fragment values, though any numerical format may be employed in the hardware graphics pipeline. Also, the sum may be carried out over multiple rendering passes. With sufficiently adaptable hardware, it may be carried out in a single rendering pass from the finest detail level to the single value.

Figure **9** shows an advanced multi-grid relaxation method **900**, in accordance with one embodiment. Abstract multi-grid relaxation approaches are well known and described generically out of the present context in: Press, William H., Teukolsky, Saul A., et. al., “Numerical Recipes in C 2nd. Ed.”, Cambridge University Press, 1992. One advantage of the present method **900** is its ability to converge much more rapidly than the brute force *Jacobi* methods of Figure **4**, where a great deal of iterative relaxation steps are required to reach a solution.

With such an approach, the solution of the partial differential equation begins at a coarse grid **902** provided by a few number of iterations of a restriction operation **906** involving an initial grid **904** of fine resolution. The discretized equation is solved on this coarse grid, as this solution may be achieved with far fewer relaxation steps than required for the fine grid. The solution may be achieved in such a way by the methods of Figure **4**. Solution values of this coarse grid **902** are then propagated via a prolongation operation **908** to generate a first finer grid **910**. The propagated values of the first finer grid **910** are not an accurate solution over the fine grid, but provide an initial distribution which typically converges to a solution after only a few more relaxation operations.

The first finer grid **910** is, in turn, solved and propagated via prolongation **908** until the desired fineness of solution **912** is achieved. By such method of working from coarse to fine grids, a solution may be found for the fine grid using far fewer relaxation operations than are required if working only with the fine grid.. Many well known configurations of restriction, coarse solution, and subsequent prolongation exist and may be found out of the present context with reference to: Press, William H., Teukolsky, Saul A., et. al., “Numerical Recipes in C 2nd. Ed.”, Cambridge University Press, 1992.

Figure **9** shows an exemplary single-ascent to the solution at fine resolution. More elaborate V-cycle and W-cycle schemes of increasing and decreasing resolution over subsequent steps are commonplace in the prior art out of the present context, and may also be implemented in a hardware graphics pipeline per the methods of the present embodiment.

Figure **10** illustrates a few grids **1000** of various resolution on which the multi-grid methods may be performed. By the method of one embodiment, the grids may exist as textures within the hardware graphics pipeline. Figure **11** illustrates simple sampling patterns **1100** for the prolongation operation **908** and restriction operation **906** as implemented in a hardware graphics pipeline, in accordance with the method of Figure **9**. As with the filtering of Figure **7A**, the prolongation and restriction operations may employ filters of any size.

The solution to partial differential equations can be used to generate 3D graphics images, utilizing the same hardware graphics pipeline that is used to find the solution to the partial differential equations. Hence, the hardware graphics pipeline can be time-shared between solving equations and rendering images.

While various embodiments have been described above, it should be understood that they have been presented by way of example only, and not limitation. Thus, the breadth and scope of a preferred embodiment should not be limited by any of the above described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.